



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO



CRYPTOGRAPHY

SESSION 4

Secret-key cryptography

Autor:

Randy Villanueva Guzmán

Profesora:

Sandra Díaz

Grupo:

3CM9

April 26, 2020

Contents

1	Module pyCryptodome	3
2	Pseudorandom generator	3
3	Key generation for secret-key	4
4	Encryption and decryption using a stream cipher	4
4.1	ChaCha20 and XChaCha20	4
4.2	Salsa20	5
5	Encryption and decryption using a block ciphers	6
5.1	AES	6
5.2	3DES	7
5.3	RC2	8
6	Modes of operation	9
6.1	ECB	10
6.2	CBC	10
6.3	CTR	11
6.4	CFB	12
6.5	OFB	12
7	Encryption and decryption combining a block cipher and each mode of operation.	13
7.1	Plaintext for CBC and CTR	13
7.2	CBC	14
7.3	CTR	17
7.4	Plaintext for CFB and OFB	18
7.5	CFB	18
7.6	OFB	20

1 Module pyCryptodome

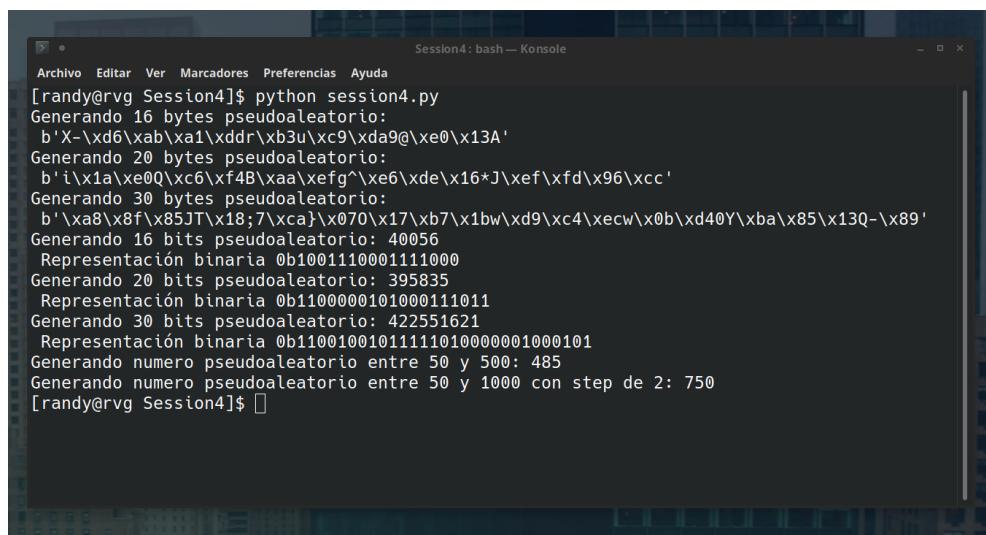
pyCryptodome es un módulo autónomo en python con primitivas criptográficas de bajo nivel. Considero que es una buena opción ya que es la mejora de otros módulos, está programado en python puro a excepción de los cifradores de bloque, donde se implementa extensiones en C para un mejor rendimiento. A diferencia de otros módulos, este implementa funciones para el manejo de padding y la generación de bytes pseudoaleatorios

2 Pseudorandom generator

Para generar números y bits de manera pseudoaleatoria *pyCryptodome* ofrece las siguientes funciones:

- `get_random_bytesN`
- `random.getrandbits(N)`
- `random.randrange([start,]stop[, step])`
- `random.randint(a, b)`
- `random.choice(seq)`
- `random.shuffle(seq)`
- `random.sample(population, k)`

Con estas funciones se pueden generar bits, numeros enteros, numeros en un rango y seleccionar dentro de una secuencia de numeros.



```

Archivo Editar Ver Marcadores Preferencias Ayuda
[randy@rvg Session4]$ python session4.py
Generando 16 bytes pseudoaleatorio:
b'X-\xd6\xab\xa1\xddr\xb3u\xc9\xda9@\xe0\x13A'
Generando 20 bytes pseudoaleatorio:
b'i\x1a\xe0Q\xc6\xf4B\xaa\xefg^xe6\xde\x16*J\xef\xfd\x96\xcc'
Generando 30 bytes pseudoaleatorio:
b'\xa8\x8f\x85JT\x18;7\xca}\x070\x17\xb7\x1bw\xd9\xc4\xecw\x0b\xd40Y\xba\x85\x13Q-\x89'
Generando 16 bits pseudoaleatorio: 40056
Representación binaria 0b1001110001111000
Generando 20 bits pseudoaleatorio: 395835
Representación binaria 0b1100000101000111011
Generando 30 bits pseudoaleatorio: 422551621
Representación binaria 0b11001001011111010000001000101
Generando numero pseudoaleatorio entre 50 y 500: 485
Generando numero pseudoaleatorio entre 50 y 1000 con step de 2: 750
[randy@rvg Session4]$ 

```

```

#Random number generation
random_16_bytes = get_random_bytes( 16 )
random_20_bytes = get_random_bytes( 20 )
random_30_bytes = get_random_bytes( 30 )
print( 'Generando 16 bytes pseudoaleatorio:\n {}' .format( random_16_bytes ) )
print( 'Generando 20 bytes pseudoaleatorio:\n {}' .format( random_20_bytes ) )
print( 'Generando 30 bytes pseudoaleatorio:\n {}' .format( random_30_bytes ) )

random_16_bits = getrandbits( 16 )
random_20_bits = getrandbits( 20 )
random_30_bits = getrandbits( 30 )
print( 'Generando 16 bits pseudoaleatorio: {} \n Representación binaria {}' )

```

```

    .format( random_16_bits , bin( random_16_bits ) ) )
print( 'Generando 20 bits pseudoaleatorio: {}\\n Representación binaria {}'
    .format( random_20_bits , bin( random_20_bits ) ) )
print( 'Generando 30 bits pseudoaleatorio: {}\\n Representación binaria {}'
    .format( random_30_bits , bin( random_30_bits ) ) )

rand_int = randint( 50 , 500 )
rand_range = randrange( 50 , 1000 , 2 )
print( 'Generando numero pseudoaleatorio entre 50 y 500: {}' .format(rand_int) )
print( 'Generando numero pseudoaleatorio entre 50 y 1000 con step de 2: {}' .format(rand_range) )

```

3 Key generation for secret-key

Para la generación de llaves se puede utilizar la función `get_random_bytes()` para obtener una llave de acuerdo al tamaño que necesitemos, por ejemplo para AES de 128,192, 256 o para DES de 168 y 112.

```

Session4: bash – Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[randy@rvg Session4]$ python session4.py
Clave AES 128 bits:
b'\x1ev~\xa5\xc6\x00\x05j \x11\x87\xd2A%*X'
Clave AES 192 bits:
b'\xe4\xfe\x90GHG\xf2\xef\x86\xc9-\xa3*/-{\'\x9a\x92j\x9f\xcbC5\x07'
Clave AES 156 bits:
b"\x03\xcd\x5\x01a\x1e\x04em\xaf\xe2'\xdf\x13k\x86\x1c\\Y\xd2^\\xba\x8c%\x86\xe8\x0c\x95\x99I"
[randy@rvg Session4]$ ■

```

```

key_Aes_128 = get_random_bytes( 16 )
key_Aes_192 = get_random_bytes( 24 )
key_Aes_256 = get_random_bytes( 32 )

print( 'Clave AES 128 bits:\\n {}' .format( key_Aes_128 ) )
print( 'Clave AES 192 bits:\\n {}' .format( key_Aes_192 ) )
print( 'Clave AES 156 bits:\\n {}' .format( key_Aes_256 ) )

```

4 Encryption and decryption using a stream cipher

4.1 ChaCha20 and XChaCha20

Es un cifrador de flujo diseñado por By Daniel J. Bernstein. La clave que utiliza es de un tamaño de 256 bits (32 bytes) y la misma clave no debe ser reutilizada en diferentes cifrados. Existen 3 variantes con clave de 8, 12 y 24 bytes.

Los textos simples y los textos cifrados (entrada / salida) solo pueden ser bytes, bytearray o memoryview. En Python 3, no puedes pasar cadenas. En Python 2, no puede pasar cadenas Unicode.

```

Session4 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[randy@rvg Session4]$ python session4.py
Cifrando con ChaCha20
Texto original: Probando cifrador con flujo, Randy Villanueva Guzman
Key: b'\xf4\x04\xaf\x91\x95\x0b\x051\x91\x01\xef\xf9\x8e\x9f4\x01.D\x02\x948r\x0cy'
Texto cifrado: GGZhdcnNj40QlwVQamBxo9T0bl/ETXjypRvyql49MzGJsyazsIkfkCk8kW7N8LqTNrnTA==
Texto decifrado: b'Probando cifrador con flujo, Randy Villanueva Guzman'
[randy@rvg Session4]$

```

```

#ChaCha20
#encrypt data
plaintext = b'Probando cifrador con flujo, Randy Villanueva Guzman'
key = get_random_bytes( 32 )
cipher = ChaCha20.new( key=key )
ciphertext = cipher.encrypt( plaintext )

nonce = b64encode(cipher.nonce).decode('utf-8')
ct = b64encode(ciphertext).decode('utf-8')
result = json.dumps({ 'nonce':nonce, 'ciphertext':ct })

print( 'Cifrando con ChaCha20' )
print( 'Texto original: {}' .format(plaintext.decode('UTF-8')) )
print( 'key: {}' .format(key) )
print( 'Texto cifrado: {}' .format(ct) )

#decrypt data
try:
    b64 = json.loads( result )
    nonce = b64decode( b64['nonce'] )
    ciphertext = b64decode( b64['ciphertext'] )
    decipher = ChaCha20.new( key=key, nonce=nonce )
    plaintext = decipher.decrypt( ciphertext )
    print( 'Texto decifrado: {}' .format(plaintext) )
except:
    print( 'Incorrect decryption' )

```

4.2 Salsa20

Cifrador de flujo diseñado por Daniel J. Bernstein. La clave de preferencia es de 256 bits, pero tambien puede trabajar con una de 128 bits.

```

Archivo Editar Ver Marcadores Preferencias Ayuda
[randy@rvg Session4]$ python session4.py
Cifrando con Salsa20
Texto original: Prueba dos, cifrando datos con Salsa20 para Crypto
key: b'\x07\x9c\xc0\xb8\x1e\x8b\xa0\xf8\xd8\xfbF\xa1}\xab\x12fw?\xeb(\x0f\x02\xf9j@&\xd1\xb0\x1e\xc3o'
Texto cifrado: b'*\x8b\xb7\x12\x9b\x0b\x0cZ\x80\x16\x07\xaa\xe8By\xac\x97\xb3\xcd,v\xd4\x92\xfa\x78\xf2\x84\xc9,\x94\x8a|\xf3p0\x98;>\xbc|\x001`\x1e<3V\xac\xd1\xc7\xb0`F\xbamG\xee'
Texto decifrado: b'Prueba dos, cifrando datos con Salsa20 para Crypto'
[randy@rvg Session4]$ 

```

```

#Salsa20
#encrypt data
plaintext = b'Prueba dos, cifrando datos con Salsa20 para Crypto'
secret = get_random_bytes( 32 )
cipher = Salsa20.new( key=secret )
msg = cipher.nonce + cipher.encrypt(plaintext)

print( 'Cifrando con Salsa20' )
print( 'Texto original: {}' .format(plaintext.decode('UTF-8')) )
print( 'key: {}' .format(secret) )
print( 'Texto cifrado: {}' .format(msg) )

#decrypt data
try:
    msg_nonce = msg[:8]
    ciphertext = msg[8:]
    cipher = Salsa20.new( key=secret, nonce=msg_nonce )
    plaintext = cipher.decrypt(ciphertext)
    print( 'Texto decifrado: {}' .format(plaintext) )
except:
    print( 'Incorrect decryption' )

```

5 Encryption and decryption using a block ciphers

5.1 AES

Es un cifrador de bloque estándar por NIST. Tiene un tamaño de bloque de datos fijo de 16 bytes. Sus claves pueden ser de 128, 192 o 256 bits de largo.

```

Session4 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[randy@rvg Session4]$ python session4.py
Cifrando con AES
Texto original: Ejemplo de cifrado con AES con clave de 126 bits
key: b'\xd3\xb5Y\xf3\x02p.\xb3=\xa13\x87\x99+T'
Texto cifrado: b'B\xf5q\xae\x04\xb9\xba\xf2\x8h\xfe\t\xb0\xbf\xb6h\xfd\x9\xc8$\x83L\xe5[\xb0|\x95\xba\x
f9\xc3\x8F?\xc0f~\xb7\xfc\x85B\xf9\x84\\xd8\xe6\x1d'
Mensaje verificado
Decifrando mensaje: b'Ejemplo de cifrado con AES con clave de 126 bits'
[randy@rvg Session4]$ 

```

#AES

```

key = get_random_bytes( 16 )
data = b'Ejemplo de cifrado con AES con clave de 126 bits'
cipher = AES.new(key, AES.MODE_EAX)
nonce = cipher.nonce
ciphertext, tag = cipher.encrypt_and_digest( data )

print( 'Cifrando con AES' )
print( 'Texto original: {}' .format(data.decode('UTF-8')) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( ciphertext ) )

decipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
plaintext = decipher.decrypt(ciphertext)
try:
    decipher.verify( tag )
    print( 'Mensaje verificado' )
    print( 'Decifrando mensaje: {}' .format( plaintext ) )
except:
    print( 'Key incorrect or message corrupted' )

```

5.2 3DES

Triple DES (o TDES o TDEA o 3DES) es un cifrado de bloque simétrico estandarizado por NIST en SP 800-67 Rev1. TDES tiene un tamaño de bloque de datos fijo de 8 bytes. Consiste en la cascada de 3 cifrados DES únicos (EDE: cifrado - descifrado - cifrado), donde cada etapa utiliza una subclave DES independiente.

```
[randy@rvg Session4]$ python session4.py
Cifrando con DES
Texto original: b'Cifrando con triple DES'
key: b'\x8aJ\x1c\x80^xb5J\xd5\x0e\xd9yI\xa7\xefp\xfb^\x0e=\x02R\xcd\xfb'
Texto cifrado: b'4S\xa4\xbf\xff\x06\xa4q\x84s\xac\xa5*T\x16o3\xc8\xb2H\xda?\xb8'
Texto decifrado: Cifrando con triple DES
[randy@rvg Session4]$
```

#3DES

```
while True:
    try:
        key_3_Des = DES3.adjust_key parity( get_random_bytes(24) )
        break
    except ValueError:
        pass

#cipher
plaintext = b'Cifrando con triple DES '
iv = get_random_bytes(8)
cipher = DES3.new( key=key_3_Des , mode=DES3.MODE_CFB , iv=iv )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key_3_Des ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key_3_Des , mode=DES3.MODE_CFB , iv=iv)
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {}' .format( decrypt_msg.decode('UTF-8') ) )
```

5.3 RC2

RC2 (Cifrado de Rivest, versión 2) es un cifrado de bloque simétrico diseñado por Ron Rivest en 1987. RC2 tiene un tamaño de bloque de datos fijo de 8 bytes. La longitud de sus claves puede variar de 8 a 128 bits. Aunque RC2 no está roto criptográficamente, no se ha analizado tan a fondo como AES, que también es más rápido que RC2.

```

Session4 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[randy@rvg Session4]$ python session4.py
Cifrando con RC2
Texto original: b'Cifrando con RC2'
key: b'\x94\xfe-\xa9\x87\x93\x1e\xc6\x84\xa1\xa0\xc6\x10\xb9\xcf'
Texto cifrado: b'B\x80\xf0d\xe0\xb0\xd7\xd1\xcf\xdd\x95;Y\x93\x8b\x01'
Texto decifrado: Cifrado con RC2
[randy@rvg Session4]$ 

```

#RC2

```

plaintext = b'Cifrando con RC2'
key = get_random_bytes( 16 )
iv = get_random_bytes( 8 )
cipher = ARC2.new( key=key , mode=ARC2.MODE_CFB , iv=iv )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con RC2' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )
decipher = ARC2.new( key=key, mode=ARC2.MODE_CFB , iv=iv )
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {} ' .format( decrypt_msg.decode('UTF-8') ) )

```

6 Modes of operation

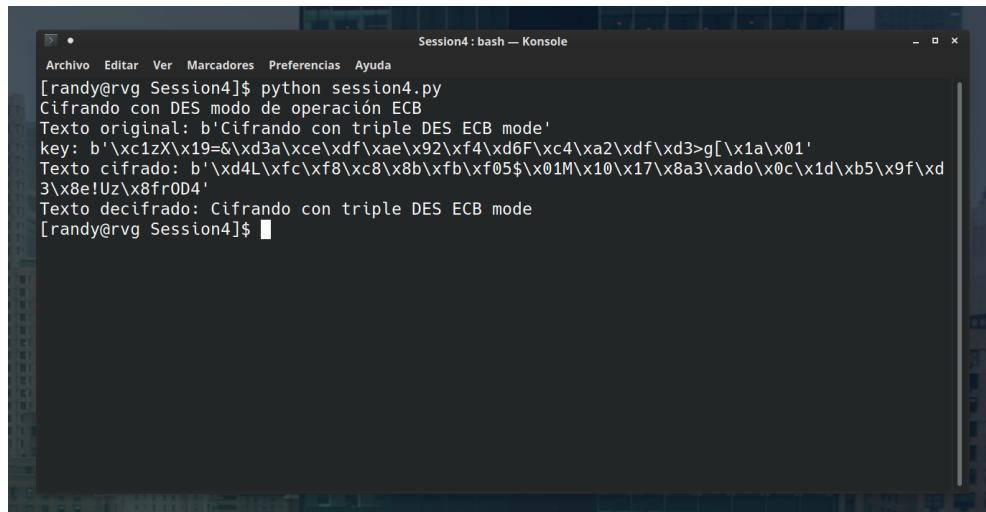
Para esta sección se generara una llave y se usara en todos los modos de operación. La llave se genera de la siguiente manera:

```

while True:
    try:
        key = DES3.adjust_key parity( get_random_bytes(24) )
        break
    except ValueError:
        pass

```

6.1 ECB



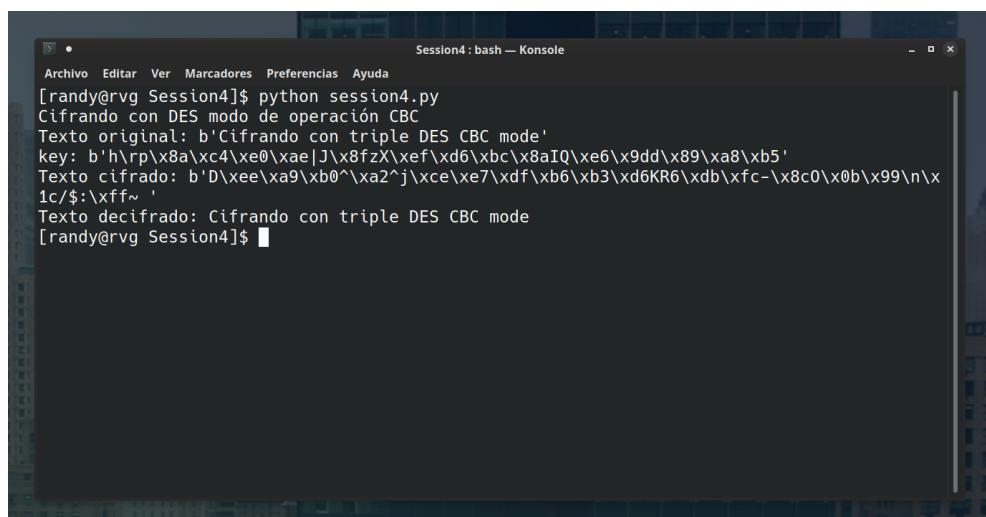
```
Session4 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[randy@rvg Session4]$ python session4.py
Cifrando con DES modo de operación ECB
Texto original: b'Cifrando con triple DES ECB mode'
key: b'\xc1z\x19=\xd3\xce\xdf\xae\x92\xf4\xd6\xc4\xa2\xdf\xd3>g[\x1a\x01'
Texto cifrado: b'\xd4L\xfc\xf8\xc8\x8b\xfb\xf0$\'\x01M\x10\x17\x8a3\xad\x0c\x1d\xb5\x9f\xd
3\x8e!Uz\x8frOD4'
Texto decifrado: Cifrando con triple DES ECB mode
[randy@rvg Session4]$
```

#ECB mode

```
plaintext = b'Cifrando con triple DES ECB mode'
cipher = DES3.new( key=key , mode=DES3.MODE_ECB )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación ECB' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_ECB )
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {} ' .format( decrypt_msg.decode('UTF-8') ) )
```

6.2 CBC



```
Session4 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[randy@rvg Session4]$ python session4.py
Cifrando con DES modo de operación CBC
Texto original: b'Cifrando con triple DES CBC mode'
key: b'h\rp\x8a\xc4\xe0\xae|J\x8fx\xef\xd6\xbc\x8aIQ\xe6\x9dd\x89\xa8\xb5'
Texto cifrado: b'D\xee\x9a\xb0^\x2^j\xce\xe7\xdf\xb6\xb3\xd6KR6\xdb\xfc-\x8c0\x0b\x99\n\x
1c/$:\xff~ '
Texto decifrado: Cifrando con triple DES CBC mode
[randy@rvg Session4]$
```

```
#CBC mode
plaintext = b'Cifrando con triple DES CBC mode'
iv = get_random_bytes(8)
cipher = DES3.new( key=key , mode=DES3.MODE_CBC , iv=iv )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación CBC' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_CBC , iv=iv )
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {}' .format( decrypt_msg.decode('UTF-8') ) )
```

6.3 CTR

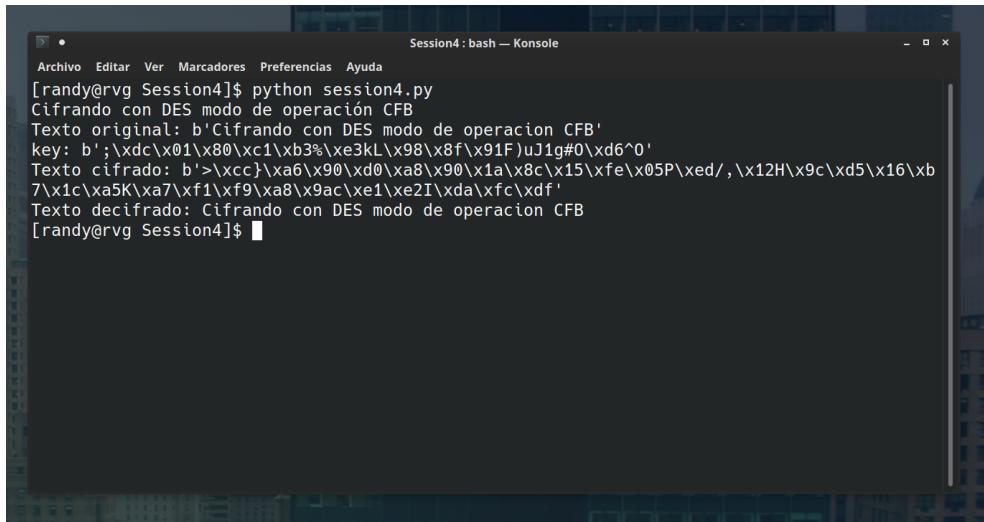
```
Session4 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
[randy@rvg Session4]$ python session4.py
Cifrando con DES modo de operación CTR
Texto original: b'Cifrando con triple DES CBC mode'
key: b'\x7f4\xf7^{\xa4\xdcQ\xc7\xe6F\xdc%\xb0p\xda8\x9F\x1a\x97\x07\xc7\x7f'
Texto cifrado: b'\xe9\n\xdf&1v\xae\xe8\x03\x18\xb8.\x8e\x98\x8f\x1a\xf6C\xc85}\xe5]\x1f\xe
7=n\xb9\x1e)\x81g'
Texto decifrado: Cifrando con triple DES CBC mode
[randy@rvg Session4]$
```

```
#CTR mode

#cipher
plaintext = b'Cifrando con triple DES CTR mode'
nonce = Random.new().read(int(DES3.block_size/2))
ctr = Counter.new(DES3.block_size*4, prefix=nonce)
cipher = DES3.new( key=key , mode=DES3.MODE_CTR , counter=ctr )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación CTR' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_CTR , counter=ctr )
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {}' .format( decrypt_msg.decode('UTF-8') ) )
```

6.4 CFB



```
[randy@rvg Session4]$ python session4.py
Cifrando con DES modo de operación CFB
Texto original: b'Cifrando con DES modo de operacion CFB'
key: b'\xd0\x01\x80\xc1\xb3%\xe3kL\x98\x8f\x91F)\u00#0\xd6^0'
Texto cifrado: b'\x00\xcc}\xa6\x90\xd0\x8a\x90\x1a\x8c\x15\xfe\x05P\xed/, \x12H\x9c\xd5\x16\xb
7\x1c\x5K\x7\xf1\xf9\x8a\x9ac\xe1\xe2I\xda\xfc\xdf'
Texto decifrado: Cifrando con DES modo de operacion CFB
[randy@rvg Session4]$
```

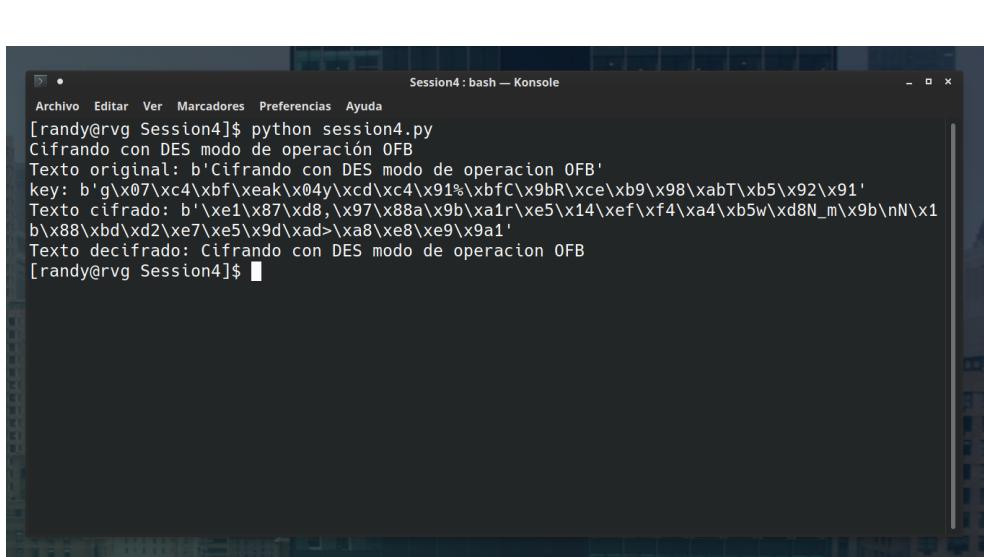
#CFB mode

#cipher

```
plaintext = b'Cifrando con DES modo de operacion CFB'
iv = get_random_bytes(8)
cipher = DES3.new( key=key , mode=DES3.MODE_CFB , iv=iv )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación CFB' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_CFB , iv=iv)
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {} ' .format( decrypt_msg.decode('UTF-8') ) )
```

6.5 OFB



```
[randy@rvg Session4]$ python session4.py
Cifrando con DES modo de operación OFB
Texto original: b'Cifrando con DES modo de operacion OFB'
key: b'g\x07\xc4\xbf\xea\x04y\xcd\xc4\x91%\xbfc\x9bR\xce\xb9\x98\xabT\xb5\x92\x91'
Texto cifrado: b'\xe1\x87\xd8,\x97\x88a\x9b\x1r\xe5\x14\xef\xf4\x4\xb5w\xd8N_m\x9b\nN\x1
b\x88\xbd\xd2\xe7\xe5\x9d\xad>\xa8\xe8\xe9\x9a1'
Texto decifrado: Cifrando con DES modo de operacion OFB
[randy@rvg Session4]$
```

```
#OFB mode

#cipher
plaintext = b'Cifrando con DES modo de operacion OFB'
iv = get_random_bytes(8)
cipher = DES3.new( key=key , mode=DES3.MODE_OFB , iv=iv )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación OFB' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_OFB , iv=iv)
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {} ' .format( decrypt_msg.decode('UTF-8') ) )
```

7 Encryption and decryption combining a block cipher and each mode of operation.

7.1 Plaintext for CBC and CTR

```
session4 > test10k.txt
Session4 > test10k.txt
1  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam in fringilla ipsum. Morbi elit elit, porta et
2
3  Proin non hendrerit sapien. Duis lectus magna, congue at faucibus sit amet, congue et ligula. Pellentesque s
4
5  Praesent eu elementum orci. Sed volutpat ante vitae justo volutpat finibus a eget quam. Donec condimentum at
6
7  Ut tempor maximus cursus. Fusce mattis, mauris sit amet convallis euismod, odio metus sodales urna, nec bibe
8
9  Curabitur faucibus feugiat augue et venenatis. Donec malesuada augue nunc, ac consectetur lacus iaculis maxim
10
11 Etiam lacinia tortor ac justo tincidunt, vitae efficitur justo consectetur. Nam gravida sit amet ex elementu
12
13 Suspendisse non enim facilisis, facilisis tellus eget, pharetra ante. Nunc vel dolor nibh. Donec at pharetra
14
15 Phasellus in blandit sapien. Praesent interdum et elit a consectetur. In hac habitasse platea dictumst. Null
16
17 Mauris id sem mattis lectus consequat pharetra. Morbi semper ex volutpat eros maximus, ac suscipit turpis ia
18
19 Nam auctor mattis purus quis rutrum. Nunc suscipit lacus odio, eu feugiat urna suscipit eu. Donec dictum fel
20
21 Vivamus laoreet feugiat nibh in tincidunt. Nullam vel condimentum urna. Aenean dictum turpis a libero hendre
22
23 Aliquam viverra massa et erat convallis commodo. Morbi lobortis erat eget felis tristique, ac tristique urna
24
25 Pellentesque mattis cursus augue eget mollis. Etiam laoreet, tellus quis pharetra vestibulum, dolor arcu mol
```

7.2 CBC

```
session4.py      test10k.txt      AES_CBC.json      Session4 > AES_CBC.json > ...
I
1   [{"iv": "18ba1+ry2iCeKHW95LH/A==", "ciphertext": "iz79KiulEiYihADCGGKFqKKJLwB7EbWt5MqdKzYiR5Zoe7RdxzGApAjKe2h6NXTYZKxR0/0mE17hbuFimX8H7VKTkp7dQ+/M/wbLQESTdU1mS514Zs42x1X0aaAYj5eqdPhsP1bRwfPztoJ0uAsQh08KGpkXswk94T+o70g1vWuRtBd02Y72tkKkrfkgMMeB8WGwbptKw65uxtdwkbaZybmLskUe5wv/ v6a0DGT1kWuRSjyo6R34sgSAjCpe0ccQciircRZ/3z0qTgyf1S9FVDQ1Q0mEsVpQMMK29S2j9nH/pP7vPE +fpd4ZJ93yab1w72MAIt593L5wPf1apL6te/guvtzIn9Dcfx5y8WbkhXwamakqipAbDRTe2TQJ4RoYz2+vFn4hV1JGBD61 +Wwk5t8EyREH/x/40/MVF/v/woac9mL8HwkuQfomSw+yL5v07alcggg8/bNiY0IC5AvzEcKpiDwGuNkpbEOuHsFaSDy1WlQWht3cYS1+UP59Msftphxk8ZN24yqDNKlvP+k +IciwozuH71m0dwbgwvBSe09r10sByBxZhPurk0/1jzNaEqwyC1P02RyPsaj0eFpZv/EIfavtelyDajtgntUucRwQmc/Dg1241mJkBePw8NgHW55vJrjBdd1d075z3kLYDxtsAaG6UzvGZP+N070xiA3gY5Qcu8cElCYf/ga/ 1kmGhgDsGnEwWbzbDzBz56aJewLzTxe1E37i6MvB4fSqDhab29w43RpkoMYP/KiTufr+DujqoVnk1ao0pg/DgMdje7wIKR/ odpkN89eCehN83tmNzNwnnon:43PDr0eKPaMgtv1os97z1 +CI9Gn1d35ZJLwi1Bn1Pm9YeYMdfPte16nGwSBu5wX1J199JGmfJebjsa6u/oA63Eqrrqv9Qp0x0kb21zAFxZ/mPuXUNhchsif5/ ad0w2e0R5Qvy4Af0pD9z182uJw9FQKqnHTTqyHehx8Dp+r3W+C0jJ/sp89AjzzJ/hhs13We8r/dYijzEhauNq85Bclg9BrfdUgjv +ZA8KeYNotFg8sos+ZDneJwCbCwycIfHMZQxW1UiwoKjIuQolak1Pra9gs81ExEMoLJt9T +GOVdcIPWXkxtswx0B93Cv7GmgLe13ta3xyJF2d6ySmHrjoxLMF9y8bqqpdn3ErVat26DHTT1ePxpd5pJutUCCk0w3JuV7k Me/bjSp0x630FE0FHeHUrJRE1yJwJyJLw5Mtztbw1L7vJsp1sPMwL61s1JhBjhqJriHw76dps70Gzuodk5gqMsBvAh3/Gcdz/ YGIK/n5ex8rcsyJL40xHem3AcuzLwaoSaKAPQ1KD60dizwbs0DNT5ASjbqMhF5zR2YAOcT1D/zJup+JtezdPdbV0y +ST+B0WB6egScRzFcYJgnVewfu70up1DmQdJL+ +Xcf0eizhnduAA0461trw14YFKd49R94M3UjwmgCAVICKj985tij1JSLNPLbnhpVm4xmmGewCwnZiMANa8kt5D96diVrnMgxSe0AF +RusPdd0a21217gZlpDw+40t9A5Mf2Jt15nSu0t1Auqj1DitwdgFuch/101G5+kNvof5vHcn6gQ +Fu1m0u8zB_PefK7x1gmDKeY9iXt0EaqyXhpCnhuaGk3r+8PjxUbhx9FfrNxN5KpiBwpvSiQz2EBj/VzEpCaDcrtXwPm53zf/ Jxcd1rHuAC17fLmIGk0+Cbfpz+TqsZkdLoakhdus56G6vNs76gbqIusRsgo/SuAw1vNvgKhtqloNjg1hL4zjnxDmv57dPsmoA +pxMeNxF4WQLM8FMs2deIMPIEdrgzQpBlzT6ND+oJ5wpGmcmcboHH1CSsuHziqndne6QnjA65Xhm/ zH4pm9fca0cmtj5K152qgh09HtkEsesVekRmuD9hns61pFw5tXuJd0212019zRwgSN,Y6m/Fks+rZevp3/6PRe8pyk7J9f7cgy +pk87wwMPRxyJNyxfsNzTp1w513C9qGuAZps4Tte8TwCJte35K20f6G56v521GgDpR3377gTMfuoyKJP
```

```
session4.py  test10k.txt  AES_CBC.json  Decrypt_AES_CBC.txt  ...
Session4 > Decrypt_AES_CBC.txt
1  |lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam in fringilla ipsum. Morbi elit elit, porta et
2
3  Proin non hendrerit sapien. Duis lectus magna, congue at faucibus sit amet, congue et ligula. Pellentesque s
4
5  Praesent eu elementum orci. Sed volutpat ante vitae justo volutpat finibus a eget quam. Donec condimentum at
6
7  Ut tempor maximus cursus. Fusce mattis, mauris sit amet convallis euismod, odio metus sodales urna, nec bibe
8
9  Curabitur faucibus feugiat augue et venenatis. Donec malesuada augue nunc, ac consectetur lacus iaculis maxi
10
11 Etiam lacinia tortor ac justo tincidunt, vitae efficitur justo consectetur. Nam gravida sit amet ex elementu
12
13 Suspendisse non enim facilisis, facilisis tellus eget, pharetra ante. Nunc vel dolor nibh. Donec at pharetra
14
15 Phasellus in blandit sapien. Praesent interdum et elit a consectetur. In hac habitasse platea dictumst. Null
16
17 Mauris id sem mattis lectus consequat pharetra. Morbi semper ex volutpat eros maximus, ac suscipit turpis ia
18
19 Nam auctor mattis purus quis rutrum. Nunc suscipit lacus odio, eu feugiat urna suscipit eu. Donec dictum fel
20
21 Vivamus laoreet feugiat nibh in tincidunt. Nullam vel condimentum urna. Aenean dictum turpis a libero hendre
22
23 Aliquam viverra massa et erat convallis commodo. Morbi lobortis erat eget felis tristique, ac tristique urna
24
25 Pellentesque mattis cursus augue eget mollis. Etiam laoreet, tellus quis pharetra vestibulum, dolor arcu mol
```

#modes of operation

```
while True:  
    try:  
        key = DES3.adjust_key parity( get_random_bytes(24) )  
        break  
    except ValueError:  
        pass
```

#ECB mode

```
plaintext = b'Cifrando con triple DES ECB mode'
cipher = DES3.new( key=key , mode=DES3.MODE_ECB )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación ECB' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
```

```

print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_ECB )
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {}' .format( decrypt_msg.decode('UTF-8') ) )

#CBC mode
plaintext = b'Cifrando con triple DES CBC mode'
iv = get_random_bytes(8)
cipher = DES3.new( key=key , mode=DES3.MODE_CBC , iv=iv )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación CBC' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_CBC , iv=iv )
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {}' .format( decrypt_msg.decode('UTF-8') ) )

#CTR mode

#cipher
plaintext = b'Cifrando con triple DES CTR mode'
nonce = Random.new().read(int(DES3.block_size/2))
ctr = Counter.new(DES3.block_size*4, prefix=nonce)
cipher = DES3.new( key=key , mode=DES3.MODE_CTR , counter=ctr )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación CTR' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_CTR , counter=ctr )
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {}' .format( decrypt_msg.decode('UTF-8') ) )

#CFB mode

#cipher
plaintext = b'Cifrando con DES modo de operacion CFB'
iv = get_random_bytes(8)
cipher = DES3.new( key=key , mode=DES3.MODE_CFB , iv=iv )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación CFB' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_CFB , iv=iv)
decrypt_msg = decipher.decrypt( msg )

```

```
print( 'Texto decifrado: {}' .format( decrypt_msg.decode('UTF-8') ) )

#OFB mode

#cipher
plaintext = b'Cifrando con DES modo de operacion OFB'
iv = get_random_bytes(8)
cipher = DES3.new( key=key , mode=DES3.MODE_OFB , iv=iv )
msg = cipher.encrypt( plaintext )
print( 'Cifrando con DES modo de operación OFB' )
print( 'Texto original: {}' .format( plaintext ) )
print( 'key: {}' .format( key ) )
print( 'Texto cifrado: {}' .format( msg ) )

#decipher
decipher = DES3.new( key=key , mode=DES3.MODE_OFB , iv=iv)
decrypt_msg = decipher.decrypt( msg )
print( 'Texto decifrado: {}' .format( decrypt_msg.decode('UTF-8') ) )

file = open( 'test10k.txt' )
text = file.read()
file.close()

text = bytes( text,"UTF-8" )

#CBC

key = get_random_bytes( 16 )
cipher = AES.new(key, AES.MODE_CBC)
ct_bytes = cipher.encrypt( pad( text, AES.block_size ) )
iv = b64encode(cipher.iv).decode('utf-8')
ct = b64encode(ct_bytes).decode('utf-8')
result = json.dumps({ 'iv':iv, 'ciphertext':ct })
file = open('AES_CBC.json','w')
file.write( result )
file.close()

file = open( 'AES_CBC.json','r' )
json_input = file.read()
file.close()

try:
    b64 = json.loads(json_input)
    iv = b64decode(b64['iv'])
    ct = b64decode(b64['ciphertext'])
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    file = open('Decrypt_AES_CBC.txt','w')
    file.write( pt.decode('UTF-8') )
    file.close()
except:
    print('Incorrect decryption CBC')
```

7.3 CTR

```
session4.py      test10k.txt      AES_CTR.json
Session4 > [AES_CTR.json] ...
1  [{"nonce": "eoJUe51Cx9A=", "ciphertext": "pjGQu5lnbFcrt3SGBHaFStthMJ61j8YNP/ G1UFyR3QoeweOftxVmU8VUcdDavQfepM7f9PYcdKLMEQLur64pY+OnGXMyhFv7Xi/Xrbhby3Y+G24UmujRKR45jytqCp +uc2KYabiLX4xotKnz7QF7q/ IbdpLqta3d5phzUMhhehLsAhzd423co6fvlxQ2BZjeUVFxUD62HTms5iNvtm734f3Z31Gr9oGHv0CR7qtgFtHOpwanX0F1WsttsawuzHuXcZ gZUjYd6xnbpcER80ugeCpxGvMS0d3wm+9yvwI+TzB2W50E1GvMSLLs/ XyMDImY1lQkP8XP7etKw5j0pykeyolJy3Ex1rq2kz8870NKDaSc00Mh1s45CSBHSCBpGdLMoExSld63kWYR1 +4B9GI9TyFb8N21iIDTnluG4417qVcBLEn57HN65Z3yAtouhcK49ens00RKpCxbubtjHPeINOrYTrvUxiflp40qSXpCkQAxHhGfrjsYuDVV1 NB6RbZVKMoaeF920E15eGKnK1Jn82JV109D+zSoBrZ6qy/Mxr1c68vEHyNfC/KN2v7xwBbSXXHJra2k9gQcmXPW/4Lrm/ s1kaykCLFVrgZmC71A8ql3PrDlyPxtytL4WPiHhWg4auXEBgfmoRxfdb5368FI0i9/ hErzt41KYacjhNs6re2NSxe4Cb9T1vow5tT0lsQHh/rfnFncx2LdIjGplHolyqm6t0RLBEYXgCqHD1gaGEhA6SyToD1cu1bz3AXK1/ gmWhs3ZxHgj10KJBfxpMc2KnWuF73s/7nlx0hIAHT3vo56llajNs/ ab5v0tthw08jGz00MPy5yl83yiqvJy3dbj8u0v0didgrZ5Gqktwzr7207Cv4CIT8ScEzpupxtbMtg0SPDrCZKT7dhZDMaUwU262hpkw5W z20j0m06vTsRvh03X61dGk1emj9bzFGzvSNQ31dy0h9UyPiw572sdvqJ3sej1ViF0Imx2MjDFJ8qMojW3iR03hje0nzuo0zlo3m82v T2kXITrzGh+GmUx5FV/ARWnckud0gWd3+twswlcQ5xUmnC6tQtibe4k1Dfze9r1CPzxLc949nuHpkAIuYyUmoTaP24H +oYGUFNKFwu9LnZkOjeS4tJaT923Xk1conljFMVCGR+rJec4Tl0rVvnDies +rQLgCzfMctp3VMRop1rTskjmT5Pj0sQxbtV66d3mVQYzehAoEUuDqxdp1hEF04H+yMUEf +DXJWGVvcRFgnRuwM6LgDbavbx2XKysvHorMRGftip0b/SBqVKH70tr3CS5giRqaNr/q+MeNYQxDftAaEGGgvI8vAimp +yIgMfs09185SvxKJtf0zF9u1zKfzAHLx6agwhFy/czC7xjKQ0hmgBywBFLp1a7s2a1w/ wUjExZCHogaF9u1zNejs41GH2620hm#N8.PXnPnTkaiXpWmf1l/5M7KYYyAs5aB2wBLYhLzUvy9VhpVwem2r9fft +mfPEHo63VzX6lnyb106TieJU0hxkd7syXY30ajqu3uiacyCwfzH2z0Vpq6hFrNM18qHH8EJFHf/ +30JyZ1qW03xliqGjywMeNWT1r1H2MnRvi0irnSluc/UltFRIV5x8AAZSVIHCJqY7qsyhlmkBjb7reco/4bwCPMgX8n/qa/ nZdlXwq4xrQ0693YgptbOTDgdkkqyZ4EExo+LVXRKyYUu243Y+ywzbtxeJEJgb4rQ+6SpMILTW +ekn4Tk3B01owoXMmJxRBW04w11stYvLPyzSdohNlw1A8n5U14+AUZ +fzKEnu2zcY0Wtqnh1FRCJ8sG0ZfxcmW4xYMvHLDG4SrbAW26M11/xQM1V2jXNAUDnkSBaqGOc8t02PpbIk2u/8EjHbrAy/ VsfgovbIMFY8diCar38HMIAR8F& nIHxR0yb1M7fv0z2U0pUi6Zsq+96EWk1zaJlh6ldqieRzuBL2NB3RxxJxbCyIb +vZCHzH7PGB3tt0Pvgpbcjrw9G8Bwwf2EKqHMDNNkyjADZlzKzX1+T3GMy1NJ2DFvMHAGPrP8BDFTM0R/k /
```

```
session4.py      Decrypt_AES_CTR.txt
Session4 > [Decrypt_AES_CTR.txt]
1  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam in fringilla ipsum. Morbi elit elit, porta et
2
3  Proin non hendrerit sapien. Duis lectus magna, congue at faucibus sit amet, congue et ligula. Pellentesque s
4
5  Praesent eu elementum orci. Sed volutpat ante vitae justo volutpat finibus a eget quam. Donec condimentum at
6
7  Ut tempor maximus cursus. Fusce mattis, mauris sit amet convallis euismod, odio metus sodales urna, nec bibi
8
9  Curabitur faucibus feugiat augue et venenatis. Donec malesuada augue nunc, ac consectetur lacus iaculis maxi
10
11 Etiam lacinia tortor ac justo tincidunt, vitae efficitur justo consectetur. Nam gravida sit amet ex elementu
12
13 Suspendisse non enim facilisis, facilisis tellus eget, pharetra ante. Nunc vel dolor nibh. Donec at pharetra
14
15 Phasellus in blandit sapien. Praesent interdum et elit a consectetur. In hac habitasse platea dictumst. Null
16
17 Mauris id sem mattis lectus consequat pharetra. Morbi semper ex volutpat eros maximus, ac suscipit turpis ia
18
19 Nam auctor mattis purus quis rutrum. Nunc suscipit lacus odio, eu feugiat urna suscipit eu. Donec dictum fel
20
21 Vivamus laoreet feugiat nibh in tincidunt. Nullam vel condimentum urna. Aenean dictum turpis a libero hendre
22
23 Aliquam viverra massa et erat convallis commodo. Morbi lobortis erat eget felis tristique, ac tristique urna
24
25 Pellentesque mattis cursus augue eget mollis. Etiam laoreet, tellus quis pharetra vestibulum, dolor arcu mol
```

```
#cipher
key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_CTR)
ct_bytes = cipher.encrypt( text )
nonce = b64encode(cipher.nonce).decode('utf-8')
ct = b64encode(ct_bytes).decode('utf-8')
result = json.dumps({'nonce':nonce, 'ciphertext':ct})

file = open('AES_CTR.json', 'w')
file.write( result )
file.close()

file = open( 'AES_CTR.json', 'r' )
json_input = file.read()
file.close()

#decipher
try:
```

```
b64 = json.loads(json_input)
nonce = b64decode(b64['nonce'])
ct = b64decode(b64['ciphertext'])
cipher = AES.new(key, AES.MODE_CTR, nonce=nonce)
pt = cipher.decrypt(ct)
file = open('Decrypt_AES_CTR.txt', 'w')
file.write(pt.decode('UTF-8'))
file.close()
except:
    print('Incorrect decryption CTR')
```

7.4 Plaintext for CFB and OFB

```
session4.py  E test15k.txt x
Session4 > E test15k.txt
1 Aenean placerat. In vulputate urna eu arcu. Aliquam erat volutpat. Suspendisse potenti. Morbi mattis felis a
2
3 Etiam posuere quam ac quam. Maecenas aliquet accumsan leo. Nullam dapibus fermentum ipsum. Etiam quis quam.
4
5 Praesent in mauris eu tortor porttitor accumsan. Mauris suscipit, ligula sit amet pharetra semper, nibh ante
6
7 Morbi leo mi, nonummy eget, tristique non, rhoncus non, leo. Nullam faucibus mi quis velit. Integer in sapien
8
9 Morbi leo mi, nonummy eget, tristique non, rhoncus non, leo. Nullam faucibus mi quis velit. Integer in sapien
10
11 Morbi leo mi, nonummy eget, tristique non, rhoncus non, leo. Nullam faucibus mi quis velit. Integer in sapien
12
13 Etiam posuere quam ac quam. Maecenas aliquet accumsan leo. Nullam dapibus fermentum ipsum. Etiam quis quam.
14
15 In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Duis sapien nunc, commodo et, interdum suscipit
16
17 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam feugiat, turpis at pulvinar vulputate, erat
18
19 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi gravida libero nec velit. Morbi scelerisque
20
21 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam feugiat, turpis at pulvinar vulputate, erat
22
23 Nam quis nulla. Integer malesuada. In enim a arcu imperdiet malesuada. Sed vel lectus. Donec odio urna, t
24
25 Praesent in mauris eu tortor porttitor accumsan. Mauris suscipit, ligula sit amet pharetra semper, nibh ante
```

7.5 CFB

```
Session4 > | AES_CFB.json > ...
  1   "iv": "c1c151xElWLRv1C1fCZ2g==", "ciphertext": "FrVzJFvhqz3Gw1n+AhEzGR9G/FhY1s+psC2d8vrZHKM
+9JG3L1wgVn2B3yLtE3jPfhTb+KasBqo32yfwmKCQ22/wN5gEWFHAGtw0giolSkImkyFC03eXV6AHDxeC+txVLseZt+wQute8o7g1/
iNCJ+vFd3yqgWimTCVCsJvMFmOs09jLBveVTxIuvOeZnyZ0mfzzw/BENFLEaG3L0coIxndNL1CvyzbQdF2AEy2b2N9S1yAkp1Nf+720k3
+ch30D0hH1ryUjk5BQ1APrugpNorL1Tp3B6w7f
ONlogAoYs1rs1rsgsekRbd6a05d5BuFh0boDwBsEvY0u1X7z10a2fxvtWeQb152i0JhdAzeCqp2wLgABVkBjz0fc70fa7
+ulPATBMM0zQDobXbn51zouEYR2D2LqPm0nHjyFvpp+YVIa+OSsxrCxNwID/kko1fwehuWS9bmnm6EittCp5Sw133j50eBhabJz3xtFT7s
+0636Vngs83+TPHCAC1ccDDQmHwlw1a13Cs1qRNFXAUshDj1b1fBofQnDy0j
ozdL55241vejSH5BjZPQNVY175hMs3aNe44Xu0zKbzgcod7wx/XfKQJ8QHeNQ2cD5CwgRsXQWh9YaPs+gDnk4MpT31LBorx6CTU/
RMsi1e1NKBg21ln3oQpxQXPHT7WzQzSu0x0eoAclLcfEqusRbwgtm15y2g
+AulJwmycqZFxGyQStgeMhi6f7GnPklxpTx1k4jOotumc1K1ogHodsXFhxz
+e0d0qJG09TCVxfugMuix1cmeHHRkjqeYsCsmxQxarALwjClyG1Ng6bv5s21SM7G7zSU9j
+jETAFtLlDobXnxFtS4fzgjFETFYR4kzB2kH827Ag3g14zRf0zKKQjSzTjlglw3fioLgqqr27Cva
+T8dpns1leVqzXV6Y1U1Ksf0w6h+DfEzwjx1OkWP1/PtB4BqBrtlcobt0JN06VwM2w0hd+Ivp5fp+30MVaae8yfLQKQzLvBZIT
+kIkBomBw/lw/F0qmX6CvwEkkokVNHFQoKjNj3UnqybvquadRj7IdUm5xnkoMt+Il55T1enEthKyhdzJ/iRBW4l
+KGjTW25Fv0Rppk1VNIckvKrkre25CuhsTo1lMXN9H8zHeyts/5C6ut00
+tXCHVs0JmrX53e5oF0P9lowlFM9yPo2j2kbsWBS5ubk3u0n0QpzJ1/D0cqeCmsGg0jp7g13wjeungvn8beNpLUxtZh0A
+mmk1TSz5eY5eYCHKMJsm/HvLwB1nNurtPwYUkqMFK41kWnf0ekxtxsJxDtJE2b9tTDKxsu/T
+vLmfvayzoxyRz8zZCPfkBdxzeAndzC8apnw/1rem1aU3/
3SAK2z7H1h1QFB8Gmd8u1N228Jg3yHSL3qjaUF0KYFwkD01bh2uZ8WUfjBh2LkDA0QvokmeVpPHPOXq85a0jdnjY1s/9et54bQd7wE
+hW115K1qf3d0mBr8F/vrK6NM2BvLwpNgJaZJPEcuyzmz88Xu4YdEY-tw+jMjYHUNHytufp5qc
+3mrh9QyaGApzHd1H1zLq1a1f5Prj3xh6d21ZemRmj+44K0F1Y290Rm4ooBRCrgin4NaGHU
+175KAfbplBwPQSKoUrbZhr30KMrGeXau/P0t59dM46Ajd4r1prARSncAdrpy0WM/UBZccGz2PFvTK0rq0/
bd19Us0o1nqWzq1xfrNb80zu0eat1kC9Dd+wi19qo/MqMjQzkn1z12t1Dy18fboltLM9WyHk+uPKYz22/RhgjyTCNyMns5/
h5wDDEDX11z5+z7o1klesu5Sp2ng8mzRYz6zAgRvEXP/
wx6G5F1TrxF8L0vS110e2rx2MLuArQrQjTmgKUksGcIN5Y470e69UnDrJzbFbdThoLPS71byKz2d652nn+Uzh4xsTuwbzKbw
+GozDfCmHmtDwnXtcs552R0dxmo9+zo0a4r6Ned0+B8t14zGKn+Uz3dXRGUjhohKcg4Pz5av90Grbk0
```

```

session4 > Decrypt_AES_CFB.txt
1 Aenean placerat. In vulputate urna eu arcu. Aliquam erat volutpat. Suspendisse potenti. Morbi mattis felis a
2
3 Etiam posuere quam ac quam. Maecenas aliquet accumsan leo. Nullam dapibus fermentum ipsum. Etiam quis quam.
4
5 Praesent in mauris eu tortor porttitor accumsan. Mauris suscipit, ligula sit amet pharetra semper, nibh ante
6
7 Morbi leo mi, nonummy eget, tristique non, rhoncus non, leo. Nullam faucibus mi quis velit. Integer in sapien
8
9 Morbi leo mi, nonummy eget, tristique non, rhoncus non, leo. Nullam faucibus mi quis velit. Integer in sapien
10
11 Morbi leo mi, nonummy eget, tristique non, rhoncus non, leo. Nullam faucibus mi quis velit. Integer in sapien
12
13 Etiam posuere quam ac quam. Maecenas aliquet accumsan leo. Nullam dapibus fermentum ipsum. Etiam quis quam.
14
15 In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Duis sapien nunc, commodo et, interdum suscipit
16
17 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam feugiat, turpis at pulvinar vulputate, erat
18
19 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi gravida libero nec velit. Morbi scelerisque
20
21 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam feugiat, turpis at pulvinar vulputate, erat
22
23 Nam quis nulla. Integer malesuada. In in enim a arcu imperdiet malesuada. Sed vel lectus. Donec odio urna, t
24
25 Praesent in mauris eu tortor porttitor accumsan. Mauris suscipit, ligula sit amet pharetra semper, nibh ante
26

```

```

key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_CFB)
ct_bytes = cipher.encrypt( text )
iv = b64encode(cipher.iv).decode('utf-8')
ct = b64encode(ct_bytes).decode('utf-8')
result = json.dumps({'iv':iv, 'ciphertext':ct})

file = open('AES_CFB.json','w')
file.write( result )
file.close()

file = open( 'AES_CFB.json','r' )
json_input = file.read()
file.close()

try:
    b64 = json.loads( json_input )
    iv = b64decode( b64['iv'] )
    ct = b64decode( b64['ciphertext'] )
    cipher = AES.new( key, AES.MODE_CFB, iv=iv )
    pt = cipher.decrypt( ct )
    file = open('Decrypt_AES_CFB.txt','w')
    file.write( pt.decode('UTF-8') )
    file.close()
except ValueError:
    print("Incorrect decryption CFB")

```

7.6 OFB

```
Session4 > {} AES_OFB.json ...
Session4 > [{"iv": "zQw5AL6dS4j3yoYhwJAxQ==", "ciphertext": "sg5767d6aCRkAFkrE5D0RCgdfMUZ+sJRQhbkdqAV24Hg7ZHI3
+ZuMyGzfA4aTmg+10SciUTR1T7vma6DRCTpZF/ius0qFP/ImMNkz/yJfHiPApS7426gMUVeMh0VZ/rQ5TEbc/XQLT5dGsAo3s4I
+2bqsuS610iWfaPR1Z8H1Rrtjt4XH/Czyxkvw0NBuFL7tAih5GSEnC++tn01GsQKxpctdGJ1SwVB9TtZKoMfQ17
+mtZe8B1xvmkp1QfhWlgVobKx91L7mvo6ohk902eUEpqgSmhr7Ma4BXgiE052Q7nj5bTEC4260085qo+8R82XuG76xYxW/
am5FV4j1rsKKX20gRLJnn5yqMcrcceMapy918X0NUXH4XFsiAokFAIT+f1aDWTPk1C2jsYot5RBtzA/
krCetcW0Y07vJreTBWVa0LTrzvh's+6zr9YJ1IILP+AEyQg+KkugYMuvZNwlGpu1QyD45nj1f/eM1Z+aJZ5Yg+zL3Y
+S9JWV2NwAu5nALVXUHtNaRb1TzK3cGbxe5CY9Cb214qhn07kQODMuAn9tpXA3KXWbNmTMAvW9VOudzA5v02yOxjqItT46W/
05AYZWHLG1b7EOuBpuWCX/Nh4TVJOXYE1Y-scfwFn2XEuPyd+E1bZzmuiHxkbucqdhs/
VVIzR0PQkIKNr6s02jzjU8Nw703j5gtszQkuM8r0PtbN1el5zbKbHx0LgnApJ7rbDsQxkAbhwNMplMB1eLeAX04HBRFcYrAd3JyYByXhcaxc
GxsH01EVzyJ2m1hpk0t
+2R1KB0sQVcb99/H92quwS01REehVVkj0KxjzKzhuBx0A2AVm19eTdV651TdkijLJWQXB1au4axMwzUvaOAasDmPeyk1Fwa/Gzwdb6eQ9m9
7phuzrh3dyCxjGCM9tQ8VbCp61s3DAm74LH1B8rrifj703TzAqYAqChbdSBjeu2roy15B+JgsSWGERoh6xubBrzNkuIyC1ZW/
M0m0nqBvkok6UZ70WAfSSHUijezpOLT7cbBLrEW4Epf0fsJjy0nZsB8CxMd95MxFxdHbt51k2//x
+oCDSiqqSKXivTt3zaFCM2CPKthVyz6qcD9+WBpWT/ZirdxMav8AI1PgaQQ7c2gZBjfTLoPjInCfnC088VtQ+L/
3wDM9ETETsDLBL1i0zdQq1hyQ/fkTFW6yZna/
JEfimYbSW0tPLLaqQb3ox1TxrouPvRq491N61Cyg0veS05r3MsV0vV2e6VFM26y1nUcwJpwC4y+359cc1Pnfwwosbc
+u53kXUAkWLXvwPvtudi/ddHHA.JfMd6u8V1x0EtQgOHG2fOfJCUL4c12WnVFea2TN+yc/
pa10yKcuXxaEfWvFbG78XB2CEC1CV4t6byJzsx6Wit35dkt31leRlp27QZIAxoiLB1zuQy/
MdowT8qloa26p3DDE6YgMIC4X5AX4AvinCXQH0vIenZuyPR/+/IpaxZGY5U2567FyJFHPrdqhrF5ItB5vFKQ0ns104np4TBao/
Xm1D1X+b66G/wlxDHmcrrP43zgkMbHid2Hvj3AHub6o4MnS72eUk1io1nHQ4MwG18I/
mwWuWRpl0ga3PMKprieS4elUpwz4uBHUGilJ_Rgt01twZimCpTiHFAeDN8x3YhK8N59CwdXg/J9cDY76+oYhMfoIHjbPjIrYw0YVtAs/
tPXWU3b8nUdiAiyqySiD062B4w/m6h3QWGxyF09g1/
EpCeQR13mdsPobDSoejKn0picZCENL1o314FQqPKN190uBuJ2Th0BoK3cStBzSktB4ySv
+rkw5CHeVc30u0ISQxmz3lWYa5g0HIMJ1zBf6jPTFJuCeX05/0ysSCVjCgHCXaubueMkh
+0uLmsW4Ad0H2DibsfeSf192wQ5Qv61o5zXNQ/1frtbpe0XXLE9IY076nf9uGG3GH1EY6QH/x7f2b6LvKvRdVSAMJBriDxVz7T7anK8S
+zswBwqzDauhMK/As1oUy3DbGIQf5508RU0Q2ttw3LxUteq/44hKyEayPqNdVtZfkJmQVUHzn/7w/
```

```
Session4 > {} Decrypt_AES_OFB.txt ...
Session4 > Etiam posuere quam ac quam. Maecenas aliquet accumsan leo. Nullam dapibus fermentum ipsum. Etiam quis quam.
Session4 > Praesent in mauris eu tortor porttitor accumsan. Mauris suscipit, ligula sit amet pharetra semper, nibh ante
Session4 > Morbi leo mi, nonummy eget, tristique non, rhoncus non, leo. Nullam faucibus mi quis velit. Integer in sapien
Session4 > Morbi leo mi, nonummy eget, tristique non, rhoncus non, leo. Nullam faucibus mi quis velit. Integer in sapien
Session4 > Etiam posuere quam ac quam. Maecenas aliquet accumsan leo. Nullam dapibus fermentum ipsum. Etiam quis quam.
Session4 > In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Duis sapien nunc, commodo et, interdum suscipit
Session4 > Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam feugiat, turpis at pulvinar vulputate, erat
Session4 > Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi gravida libero nec velit. Morbi scelerisque
Session4 > Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam feugiat, turpis at pulvinar vulputate, erat
Session4 > Nam quis nulla. Integer malesuada. In in enim a arcu imperdiet malesuada. Sed vel lectus. Donec odio urna, t
Session4 > Praesent in mauris eu tortor porttitor accumsan. Mauris suscipit, ligula sit amet pharetra semper, nibh ante
```

```
key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_OFB)
ct_bytes = cipher.encrypt(text)
iv = b64encode(cipher.iv).decode('utf-8')
ct = b64encode(ct_bytes).decode('utf-8')
result = json.dumps({'iv':iv, 'ciphertext':ct})

file = open('AES_OFB.json', 'w')
file.write(result)
file.close()

file = open('AES_OFB.json', 'r')
json_input = file.read()
file.close()

try:
    b64 = json.loads(json_input)
    iv = b64decode(b64['iv'])
```

```
ct = b64decode(b64['ciphertext'])
cipher = AES.new(key, AES.MODE_OFB, iv=iv)
pt = cipher.decrypt(ct)
file = open('Decrypt_AES_OFB.txt', 'w')
file.write(pt.decode('UTF-8'))
file.close()
except ValueError:
    print("Incorrect decryption OFB")
```